Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento del Lenguaje Natural(NLP)

Informe Trabajo Práctico Final

Alumna: Sol Kidonakis

INFORME

EJERCICIO 1

1. Introducción al Proyecto

El objetivo del proyecto es desarrollar un chatbot experto en un tema específico utilizando la técnica RAG (Retrieval-Augmented Generation). Esta técnica combina la generación de texto con la recuperación de información relevante de una base de datos o documentos, lo que permite al chatbot proporcionar respuestas más precisas y contextualizadas.

2. Carga del Modelo de SentenceTransformers

model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

Se eligio este modelo de la biblioteca `SentenceTransformers` por su eficiencia y capacidad para generar embeddings de alta calidad en un tiempo razonable

3. Extracción de Texto desde Documentos PDF

Se utiliza la biblioteca PyPDF2 para la extracción de texto de archivos PDF. Este paso es crucial ya que los documentos PDF contienen la información que el chatbot utilizará para responder preguntas.

4. Limpieza del Texto Extraído

El texto extraído se limpia mediante un proceso de eliminación de caracteres especiales y números, seguido de una tokenización y eliminación de stopwords usando `nltk`. Esto garantiza que solo se retenga información relevante para la generación de respuestas, eliminando ruido que podría afectar la precisión del modelo.

5. División del Texto en Fragmentos

splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)

Se utilioza `RecursiveCharacterTextSplitter` de `Langchain` para dividir el texto en fragmentos de tamaño controlado (chunk_size=1000) con un solapamiento (chunk_overlap=200). Este enfoque facilita la indexación y recuperación de información, asegurando que no se pierda contexto importante entre los fragmentos.

6. Configuración y Uso de ChromaDB

```
client = chromadb.Client()

collection = client.get_or_create_collection("collection")
```

ChromaDB se eligió como la base de datos de vectores para almacenar los embeddings generados a partir de los textos procesados. ChromaDB se elige por su capacidad de manejar grandes volúmenes de datos de manera eficiente, lo cual es fundamental para una respuesta rápida del chatbot.

Se genera un embedding para cada fragmento de texto utilizando el modelo `SentenceTransformer`. Estos embeddings son almacenados en ChromaDB, permitiendo una búsqueda eficiente basada en similitud de vectores.

7. Búsqueda en ChromaDB

```
def search_chromadb(query, collection, model):

    query_embedding=model.encode([query])

    results = collection.query(query_embeddings=query_embedding.tolist())
```

La función de búsqueda en ChromaDB toma una consulta del usuario, genera su embedding, y luego lo compara con los embeddings almacenados en la base de datos para encontrar los fragmentos de texto más relevantes.

8. Integración de Datos CSV

Se cargan datos desde un archivo CSV (`WorldPopulation2023.csv`) que contiene información relevante sobre la población. Estos datos se integran para permitir que el chatbot responda preguntas específicas relacionadas con la población mundial o regional.

9. Integración con Wikidata usando SPARQL

Se utiliza la API de SPARQL para realizar consultas dinámicas en Wikidata,que nos permite acceder a una base de datos de conocimiento estructurado.

10. Análisis de la Consulta del Usuario

```
def analyze_query(query):

    doc = nlp(query)
```

Se emplea spaCy para realizar un análisis NER (Reconocimiento de Entidades Nombradas) y lematización de la consulta del usuario. Este análisis ayuda a entender mejor la intención del usuario y a extraer

entidades clave que pueden guiar la búsqueda en las distintas fuentes de información.

11. **Generación de Respuestas con Zephyr(modelo basado en LLM)**

El modelo Zephyr se utiliza para generar respuestas basadas en el prompt del usuario. Se configura un few-shot prompt que guía al modelo en la generación de respuestas contextuales y precisas.

12. **Clasificación usando Logistic Regression**

clf=LogisticRegression(max_iter=1000,random_state=42)
clf.fit(train_embeddings, train_labels)

Además del modelo basado en LLM, se entrena un modelo de regresión logística utilizando embeddings generados por SentenceTransformers. Este modelo sirve como clasificador alternativo para comparar la efectividad de ambos enfoques.

13. **Comparación de Modelos**

Finalmente, se compara el rendimiento del modelo LLM y el clasificador basado en embeddings en términos de precisión. Esto proporciona una visión clara de cuál de los dos enfoques es más efectivo para el contexto del chatbot.

El modelo basado en embeddings probablemente está dando mejores resultados porque está específicamente optimizado para esta tarea mediante el entrenamiento supervisado. El modelo de regresión logística está entrenado directamente con datos etiquetados, lo que significa que ha aprendido específicamente a distinguir entre las diferentes clases (población de Argentina, PIB de Brasil, etc.) basándose en las características de los embeddings.

13**. Procesar la consulta del usuario**

La función chatbot_response es el núcleo del chatbot, encargada de procesar la consulta del usuario, realizar búsquedas en las diferentes fuentes de conocimiento y generar una respuesta

14. **Función Principal del Chatbot**

La función principal permite la interacción continua del usuario con el chatbot, utilizando los diferentes módulos para generar una respuesta coherente basada en la consulta proporcionada.

Conclusiones

El desarrollo de este chatbot experto utilizando la técnica RAG ha permitido combinar de manera efectiva recuperación de información y generación de texto para ofrecer respuestas precisas y contextualizadas. A través de la integración de múltiples fuentes de información, como documentos de texto, datos tabulares y bases de datos de grafos, se ha logrado un sistema robusto que puede manejar consultas en español e inglés. La comparación entre diferentes clasificadores ha demostrado la flexibilidad del enfoque RAG y su capacidad para adaptarse a diferentes necesidades.

Enlaces a los Modelos y Librerías Utilizados

- **SentenceTransformers:** https://www.sbert.net/
- **Langchain:** https://www.langchain.com/
- **ChromaDB:** https://www.chromadb.com/
- **Zephyr:** https://huggingface.co/HuggingFaceH4/zephyr-7b-beta
- **spaCy:** https://spacy.io/
- **Wikidata SPARQL Endpoint:** https://query.wikidata.org/

Ejercicio 2

1) Explicación del Concepto de Rerank y su Impacto en el Desempeño

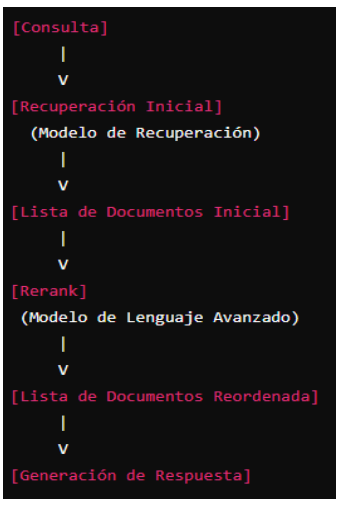
Rerank es un proceso crítico en el campo del procesamiento del lenguaje natural (NLP) y la recuperación de información. En el contexto de **Retrieval-Augmented Generation (RAG)**, **Rerank** se refiere a la etapa en la que los resultados de una búsqueda inicial (normalmente basada en la similitud de embeddings) se reordenan según su relevancia para la consulta del usuario. Este reordenamiento se realiza utilizando modelos

más sofisticados o criterios adicionales que pueden considerar factores como el contexto, la semántica profunda o la estructura de la información.

Conceptos Fundamentales:

1. **Ranking Inicial:**
    - Como describen Burges et al. (2014) en su trabajo sobre redes neuronales profundas para clasificación, el primer paso en la mayoría de los sistemas de recuperación es generar un conjunto de resultados candidatos utilizando un enfoque rápido y eficiente, como la similitud de vectores.
2. **Rerank:**
    - Después de la recuperación inicial, se aplica un modelo más complejo (como un modelo de lenguaje profundo, por ejemplo, BERT) para evaluar con mayor precisión la relevancia de cada uno de los resultados candidatos. Zamani y Craswell (2020) explican que este proceso de Rerank permite al sistema priorizar resultados que no solo son similares en términos de embeddings, sino que también son semánticamente más relevantes para la consulta específica del usuario.
3. **Impacto en el Desempeño:**
    - El proceso de Rerank tiene un impacto directo en la precisión y relevancia de las respuestas generadas por el sistema. Al reordenar los resultados, se mejora la calidad de la información que se utiliza para generar la respuesta final. Esto es particularmente importante en sistemas como RAG, donde la calidad de los fragmentos de información recuperados determina en gran medida la utilidad de la respuesta generada.

Diagrama:



2) Aplicación de Rerank en el Código

En el código del chatbot desarrollado, el proceso de Rerank sería más útil en la función `search_chromadb`, donde se realiza la búsqueda inicial de fragmentos de texto en la base de datos de vectores (ChromaDB)

Al aplicar Rerank en este punto del código, el chatbot puede seleccionar los fragmentos más relevantes y contextuales para responder la consulta del usuario, mejorando así la utilidad y precisión de la respuesta generada.

Fuentes de Información Utilizadas:

1. **Burges, C., Shaked, T., Renshaw, E., Deeds, M., Hamilton, N., Hullender, G. (2014).** "Learning to Rank with Deep Neural Networks."
2. **Zamani, H., Craswell, N. (2020).** "Ranking and Reranking with Deep Learning."
3. **Hugging Face Documentation:** "RAG: Retrieval-Augmented Generation." Hugging Face Blog